Universal Content Launcher

A series of tutorials detailing the creation of a content launcher in Microsoft Visual C#

Advanced Tutorial Two

Steam Integration

Part A

An Introduction to Steam Integration

Contents

Introduction **3**
Tutorial Requirements 3
Part One: Introducing the Steam API **4**
Part Two: Reading Steam Information **11**

Introduction

In our series of basic tutorials, we embarked on a journey to create a basic content launcher in C#. We've also looked briefly at the Windows API, and how we can call external functions from our own code.

Now it's time to get serious. Let's look at how we can integrate our launcher into the Steam API to allow us to process available Steam content.

Tutorial Requirements

The advanced tutorials are enhancements to the first five basic tutorials that built our Universal Content Launcher. You can either work through these tutorials first, or download the source code, and have a go at this advanced tutorial.

For the purposes of this tutorial, it is assumed that you fully understand everything that is in the downloadable source code.

Part One: Introducing the Steam API

In our first advanced tutorial, we took a look at the windows API, and how we can call functions from an external dll file to do a named task, specifically we were extracting an icon from an executable file. We also looked at how we could define type structures for objects that the external functions use, and briefly glanced over Marshalling operations for external calls to unmanaged code.

Interfacing to Steam is pretty much the same, there are certain types we need to define, such as a Steam error object, or a Steam application object, and there are a lot of functions that are available for us to do various things with the Steam content.

These functions were originally written for the interface between the Steam user interface, the Steam configuration files and the Steam network itself, and as such we can use these to communicate with Steam in exactly the same way that its own user interface does.

This is all good, and you could of course write your own launcher to interface with a legitimate steam.dll file and login to Steam through your own valid accounts, but that is not the objective of this tutorial. This tutorial is designed to show the usefulness, and completeness of some of the emulators that are currently in circulation.

These emulators allow you to interface with Steam in the same way that you would do with a legitimate steam.dll file, but without ever communicating with the Steam network itself. This would therefore give you a complete offline solution for playing Steam games on your own computer in single player mode, or even via a LAN for some multiplayer goodness, a far more desirable proposition than logging in to Steam whenever you want to do this.

At the time of writing, I am familiar with one emulator that does a complete job of emulating the steam.dll file that contains the functions we are going to use. This is, of course, SteamEmu by hCUPa, and it is worth pointing out that hCUPa not only emulates Steam functions, bit gives developers some functions of their own, which we can use to enhance our launcher experience.

There may well be other emulators that are complete, and I know of some that are nearing completion, but for the purpose of this tutorial we will stick with what is currently working, and address the rest as and when they surface.

So, how do we use SteamEmu in our launcher?

Well, to start with let's talk a little about how it works. SteamEmu in its current form relies on a file called clientregistry.blob to read important information related to the content that is available on Steam, amongst other things. This is no different from the original Steam UI itself, and indeed we can use the same configuration file for SteamEmu. The file itself is created by the original Steam client, and is updated by Valve on a regular basis whenever they add new content to the Steam network. Communicating with this file allows us to avoid having to create lengthy and messy configuration files for all of the Steam content that we want to process, but locks us into having to keep an up to date clientregistry.blob for our launcher to be compatible with all the very latest Steam content.

So, does SteamEmu allow us to directly read the content of this file? No it doesn't, not in the way that you are probably thinking, but what it does do is allow us to perform certain functions based on the data in this file. SteamEmu essentially does all the hard work for us, and unless stated otherwise, these are not exclusive functions implemented by hCUPa, but functions that are also available in a legitimate steam.dll.

This means that we could write a single piece of code that could either use a legitimate steam account, or the emulator, simply by swapping the steam.dll file that our launcher is currently using. Don't get too excited though, as this tutorial is only going to cover the basics of running the emulator. There is a bit more work that you would need to do to write a legitimate launcher.

So, let's look at some types and functions that we are going to be using throughout this tutorial. I will give you a brief description of what they do, and we'll cover them in more detail as and when we come across them.

First, in my UCL project, I have created a new class file for my Steam definitions called UCLSteamEmu.cs and to start with, added the following code:

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;

namespace UCL
{
    public class UCLSteamEmu
    {

        public enum eSteamError
        {
            eSteamErrorNone = 0,
            eSteamErrorUnknown = 1,
            eSteamErrorLibraryNotInitialized = 2,
            eSteamErrorLibraryAlreadyInitialized = 3,
            eSteamErrorConfig = 4,
            eSteamErrorContentServerConnect = 5,
            eSteamErrorBadHandle = 6,
            eSteamErrorHandlesExhausted = 7,
            eSteamErrorBadArg = 8,
            eSteamErrorNotFound = 9,
            eSteamErrorRead = 10,
            eSteamErrorEOF = 11,
            eSteamErrorSeek = 12,
            eSteamErrorCannotWriteNonUserConfigFile = 13,
            eSteamErrorCacheOpen = 14,
            eSteamErrorCacheRead = 15,
            eSteamErrorCacheCorrupted = 16,
            eSteamErrorCacheWrite = 17,
            eSteamErrorCacheSession = 18,
            eSteamErrorCacheInternal = 19,
            eSteamErrorCacheBadApp = 20,
            eSteamErrorCacheVersion = 21,
            eSteamErrorCacheBadFingerPrint = 22,
            eSteamErrorNotFinishedProcessing = 23,
            eSteamErrorNothingToDo = 24,
            eSteamErrorCorruptEncryptedUserIDTicket = 25,
            eSteamErrorSocketLibraryNotInitialized = 26,
            eSteamErrorFailedToConnectToUserIDTicketValidationServer =
27,
            eSteamErrorBadProtocolVersion = 28,
            eSteamErrorReplayedUserIDTicketFromClient = 29,
            eSteamErrorReceiveResultBufferTooSmall = 30,
            eSteamErrorSendFailed = 31,
            eSteamErrorReceiveFailed = 32,
            eSteamErrorReplayedReplyFromUserIDTicketValidationServer =
33,
            eSteamErrorBadSignatureFromUserIDTicketValidationServer =
34,
            eSteamErrorValidationStalledSoAborted = 35,
            eSteamErrorInvalidUserIDTicket = 36,
            eSteamErrorClientLoginRateTooHigh = 37,
            eSteamErrorClientWasNeverValidated = 38,
            eSteamErrorInternalSendBufferTooSmall = 39,
            eSteamErrorInternalReceiveBufferTooSmall = 40,
            eSteamErrorUserTicketExpired = 41,
```

```
            eSteamErrorCDKeyAlreadyInUseOnAnotherClient = 42,
            eSteamErrorNotLoggedIn = 101,
            eSteamErrorAlreadyExists = 102,
            eSteamErrorAlreadySubscribed = 103,
            eSteamErrorNotSubscribed = 104,
            eSteamErrorAccessDenied = 105,
            eSteamErrorFailedToCreateCacheFile = 106,
            eSteamErrorCallStalledSoAborted = 107,
            eSteamErrorEngineNotRunning = 108,
            eSteamErrorEngineConnectionLost = 109,
            eSteamErrorLoginFailed = 110,
            eSteamErrorAccountPending = 111,
            eSteamErrorCacheWasMissingRetry = 112,
            eSteamErrorLocalTimeIncorrect = 113,
            eSteamErrorNetwork = 200
        }

        public enum eDetailedPlatformErrorType
        {
            eNoDetailedErrorAvailable,
            eStandardCerrno,
            eWin32LastError,
            eWinSockLastError,
            eDetailedPlatformErrorCount
        }

        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
Pack = 4)]
        public struct TSteamError
        {
            public eSteamError eSteamError;
            public eDetailedPlatformErrorType eDetailedErrorType;
            public uint ErrCode;
            [MarshalAs(UnmanagedType.ByValTStr, SizeConst = 255)]
            public string ErrDescription;
        }


        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
Pack = 4)]
        public struct TSteamApp
        {
            public string Name;
            public uint maxNameChars;
            public string latestVersionLabel;
            public uint maxLatestVersionLabelChars;
            public string currentVersionLabel;
            public uint maxCurrentVersionLabelChars;
            public string cacheFile;
            public uint maxCacheFileChars;
            public uint id;
            public uint latestVersionId;
            public uint currentVersionId;
            public uint minCacheFileSizeMB;
            public uint maxCacheFileSizeMB;
            public uint numLaunchOptions;
            public uint numIcons;
```

```
            public uint numVersions;
            public uint numDependencies;
        }

        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
Pack = 4)]
        public struct TSteamAppLaunchOption
        {
            public string description;
            public uint maxDescChars;
            public string cmdLine;
            public uint maxCmdLineChars;
            public uint index;
            public uint iconIndex;
            public uint noDesktopShortcut;
            public uint noStartMenuShortcut;
            public uint isLongRunningUnattended;
        }

        [StructLayout(LayoutKind.Sequential, CharSet = CharSet.Ansi,
Pack = 4)]
        public struct TSteamAppStats
        {
            public uint NumApps;
            public uint MaxNameChars;
            public uint MaxVersionLabelChars;
            public uint MaxLaunchOptions;
            public uint MaxLaunchOptionDescChars;
            public uint MaxLaunchOptionCmdLineChars;
            public uint MaxNumIcons;
            public uint MaxIconSize;
        }

        [DllImport("steam.dll")]
        public static extern int SteamSetUser(string User, ref uint
UserSet, ref TSteamError SteamErr);

        [DllImport("steam.dll")]
        public static extern int SteamGetAppStats(ref TSteamAppStats
AppStats, ref TSteamError SteamErr);

        [DllImport("steam.dll")]
        public static extern int SteamGetAppIds(uint[] AppIDs, uint
MaxAppIDs, ref TSteamError SteamErr);

        [DllImport("steam.dll")]
        public static extern int SteamIsAppSubscribed(uint AppID, ref
uint IsAppSubscribed, ref uint IsAppReady, ref TSteamError SteamErr);

        [DllImport("steam.dll")]
        public static extern int SteamEnumerateApp(uint AppID, ref
TSteamApp AppStruc, ref TSteamError SteamErr);

        [DllImport("steam.dll")]
        public static extern int SteamEnumerateAppLaunchOption(uint
Appid, uint LaunchIdx, ref TSteamAppLaunchOption SteamAppLaunchOption,
ref TSteamError SteamErr);
```

```
        [DllImport("steam.dll")]
        public static extern int SteamGetAppDir(uint AppId,
StringBuilder Directory, ref TSteamError SteamErr);

        [DllImport("steam.dll")]
        public static extern int SteamEmu_GetGcfList(uint AppID,
StringBuilder Buf, uint BufSize);

    }
}
```

Those are the minimum types and functions we are going to use in order to read and process our Steam content, we will be adding to them as we go through the tutorials, but here's a brief description of what we have at the moment:

`TsteamError`

Most steam function calls will return an error object, which consists of an error object definition `TSteamError` and two enum definitions for basic and detailed platform error codes which are defined in the following enums:

`eSteamError`

`eDetailedPlatformErrorType`

Now we have a basic model for reporting back errors that may occur when we call an external Steam function, we can look at what functions we will need to utilise to actually include Steam content in our launcher.

`SteamSetUser`
First, whatever we do in steam we need to do it for a specified user, now for legitimate Steam you would need to login and process the login correctly, but for the purpose of an emulator, we just need to set the Steam user which will then be used by the internal functions throughout Steam. If we don't set any user, then the emulator will assume a default name, but for the purposes of this tutorial we will, of course, be using the user "UCL".

`SteamGetAppStats`
This function returns a steam application statistics object `TSteamAppStats` which details information about the applications currently stored in the clientregistry.blob file. We will be using it to determine the total number of applications that are reported for the current clientregistry.blob

SteamGetAppIds
We're going to use this function to retrieve an array of all the available Application IDs that are in the clientregistry.blob file once we know how many of them to process with `SteamGetAppStats`. We can then enumerate the returned array, and process each AppID in turn.

SteamIsAppSubscribed
This function returns the subscription status of a steam application by AppID and reports back whether or not the current Steam account actually owns the application. An emulator will, of course, always return that we do own the application, or this would be a very short tutorial!

SteamEnumerateApp
This function returns a steam application object `TSteamApp` for the given AppID. This reports back information like the Name of the application, its cachefile and how many dependencies it has.

SteamEnumerateAppLaunchOption
This function returns a launch option object `TSteamAppLaunchOption` for the given AppID. This reports back information like the command line we need to execute to launch the application.

SteamGetAppDir
This function returns the installation directory of the given AppID, taking into account the current Steam user that is set. So if my SteamApps folder was:

C:\Games\Steam\SteamApps

And the AppID in question was 70 (Half-Life) and I was logged in as the user "UCL" then this would return:

C:\Games\Steam\SteamApps\UCL\Half-Life\

`SteamEmu_GetGcfList`
This is our first function written exclusively by hCUPa for developers, it is not available in the legitimate steam.dll. This function will return a string detailing all the GCFs required for the AppID specified. The string is delimited by semi colons and has a special use in that files proceeded with a "@" symbol indicate that they are optional and therefore not required to launch the content.

Using the functions and types listed above, we have all we need to enumerate all the steam apps in our clientregistry.blob file, we won't go much further than this in the first tutorial on Steam integration, so let's knock up a simple method in UCL that enumerates a clientregistry.blob file by using SteamEmus steam.dll and outputs some content information to a text file.

Part Two: Reading Steam Information

First, we need the SteamEmu files to integrate with, so download the latest version of these files from the cs.rin.ru forum in the SteamEmu thread and copy them into your UCL project folder. You need to have these three files:

1. Steam.dll
2. SteamEmu.ini
3. Clientregistry.blob

Note, you will have to manually alter the SteamEmu.ini so that the GCF file location is correct for your machine, so if all your GCF files are located in:

```
C:\Games\Steam\SteamApps
```

Then your SteamEmu.ini file should read like this:

```
[Emulator]
BaseDir=C:\Games\Steam\SteamApps
PreferLocal=1
OverwriteLocalCopy=0
```

Also, don't forget to set all three files "Copy to Output Directory" property to "Copy if Newer" in the solution explorer, now you're all set. Note, if you don't see the files in the solution explorer after copying them in to your project folder, then click on the "Show All Files" button at the top of the solution explorer and then right click on each of the files and select "Include in Project".

Now, let's make sure our new method will be called when UCL starts by modifying the `UCLMain_Load` method in `UCLMain.cs` as shown:

```
private void UCLMain_Load(object sender, EventArgs e)
{
    LoadSteamContentList();

    LoadContentList();
}
```

Now let's write the method, here it is in all its glory, and we'll step through it line by line below:

```
        public void LoadSteamContentList()
        {

            UCLSteamEmu.TSteamError SErr = new
UCLSteamEmu.TSteamError();
            uint iUserSet = 0;
            int iSteamReturn = 0;

            iSteamReturn = UCLSteamEmu.SteamSetUser("UCL", ref
iUserSet, ref SErr);

            if (iUserSet == 1)
            {

                UCLSteamEmu.TSteamAppStats AppStats = new
UCLSteamEmu.TSteamAppStats();

                iSteamReturn = UCLSteamEmu.SteamGetAppStats(ref
AppStats, ref SErr);

                if (AppStats.NumApps > 0)
                {

                    FileInfo LogFile = new FileInfo("SteamApps.txt");
                    StreamWriter TextLogFile = LogFile.CreateText();

                    uint []RawAppIDs;
                    RawAppIDs = new uint[AppStats.NumApps];
                    iSteamReturn =
UCLSteamEmu.SteamGetAppIds(RawAppIDs, AppStats.NumApps, ref SErr);

                    uint iBuffSize = 512;

                    foreach (uint RawAppID in RawAppIDs)
                    {

                        uint iIsAppSubscribed = 0;
                        uint iIsAppReady = 0;
                        iSteamReturn =
UCLSteamEmu.SteamIsAppSubscribed(RawAppID, ref iIsAppSubscribed, ref
iIsAppReady, ref SErr);

                        UCLSteamEmu.TSteamApp AppNFO = new
UCLSteamEmu.TSteamApp();

                        AppNFO.Name = new String(' ', 255);
                        AppNFO.maxNameChars = 255;
                        AppNFO.currentVersionLabel = new String(' ',
255);
                        AppNFO.maxCurrentVersionLabelChars = 255;
                        AppNFO.latestVersionLabel = new String(' ',
255);
                        AppNFO.maxLatestVersionLabelChars = 255;
                        AppNFO.cacheFile = new String(' ', 255);
                        AppNFO.maxCacheFileChars = 255;
```

```
                        iSteamReturn =
UCLSteamEmu.SteamEnumerateApp(RawAppID, ref AppNFO, ref SErr);

                        if (AppNFO.numDependencies != 0
                        {

                            UCLSteamEmu.TSteamAppLaunchOption AppLO =
new UCLSteamEmu.TSteamAppLaunchOption();
                            AppLO.description = new String(' ', 255);
                            AppLO.maxDescChars = 255;
                            AppLO.cmdLine = new String(' ', 255);
                            AppLO.maxCmdLineChars = 255;
                            AppLO.index = 0;
                            AppLO.iconIndex = 0;
                            AppLO.noDesktopShortcut = 0;
                            AppLO.noStartMenuShortcut = 0;
                            AppLO.isLongRunningUnattended = 0;

                            iSteamReturn =
UCLSteamEmu.SteamEnumerateAppLaunchOption(RawAppID, 0, ref AppLO, ref
SErr);

                            StringBuilder strAppDir = new
StringBuilder((int)iBuffSize);
                            iSteamReturn =
UCLSteamEmu.SteamGetAppDir(RawAppID, strAppDir, ref SErr);

                            StringBuilder GCFList = new
StringBuilder((int)iBuffSize);
                            iSteamReturn =
UCLSteamEmu.SteamEmu_GetGcfList(RawAppID, GCFList, iBuffSize);

                            TextLogFile.WriteLine("Steam AppID:      "
+ RawAppID);
                            TextLogFile.WriteLine("      Subscribed: "
+ iIsAppSubscribed);
                            TextLogFile.WriteLine("      Name:       "
+ AppNFO.Name);
                            TextLogFile.WriteLine("      Path:       "
+ strAppDir.ToString());
                            TextLogFile.WriteLine("      cmdLine:    "
+ AppLO.cmdLine);
                            TextLogFile.WriteLine("      GCFList:    "
+ GCFList.ToString());
                            TextLogFile.Write(TextLogFile.NewLine);

                        }

                    }
                    TextLogFile.Close() ;

                }
                else
                {

                    MessageBox.Show("Could not find any Steam content
to load!");
```

```
            }
        }else{
            MessageBox.Show("Could not set the Steam User!");
        }
    }
```

Wow, big method, let's break it up and explain what is happening:

First, I would like to bring to your attention something that may cause you much grief if you are using a 64 bit development environment, if you are not, then you may want to make a note of this for the future!

So we're talking about a situation where you are using Visual Studio 2005 in Windows XP x64 or Windows 2003 x64 or even Windows Vista x64. In this instance, you need to make sure that your program properties are set to compile to an x86 (32bit) executable, as we are directly interfacing with a 32bit dll. If you don't do this, then you will receive a bad image error whenever the external dll is accessed by your code. To do this go to Project > Properties and make sure the target platform is set to x86 and not x64 or "Any CPU".

Ok, let's explain the code:

```
            UCLSteamEmu.TSteamError SErr = new
UCLSteamEmu.TSteamError();
```

First we create a new `TSteamError` object to hold error data should any of our function calls fail. It is preceded by `UCLSteamEmu` as that is what we called our SteamEmu class.

```
            uint iUserSet = 0;
            int iSteamReturn = 0;
```

Now we initialise two variables of the correct types to be used in the next function call.

```
            iSteamReturn = UCLSteamEmu.SteamSetUser("UCL", ref
iUserSet, ref SErr);
```

Here we do the call to set the Steam user to "UCL". Notice that we use `iSteamReturn` to hold the return value of the function, which could be used to determine if it succeeded or not and output the contents of the `SErr` object if need be. Within the function itself we pass the string "UCL" and the variable `iUserSet` along with the error object `SErr`. The important thing here is that `iUserSet` and `SErr` are both passed by reference, so their values can be modified by the external function.

```
            if (iUserSet == 1)
             {
```

Indeed, if the function was succesfull, then `iUserSet` will be set by `SteamSetUser` to the value 1, so we can check for this before continuing.

```
                UCLSteamEmu.TSteamAppStats AppStats = new
UCLSteamEmu.TSteamAppStats();
```

Next we define an object `TSteamAppStats` to retrieve the number of applications in our clientregistry.blob file.

```
                iSteamReturn = UCLSteamEmu.SteamGetAppStats(ref
AppStats, ref SErr);
```

And here we have made the call to `SteamGetAppStats`.

```
                if (AppStats.NumApps > 0)
                {
```

`AppStats` was passed by reference, and has therefore been populated by the external function, so if we have more than one application in our clientregistry.blob file, then let's continue.

```
                    FileInfo LogFile = new FileInfo("SteamApps.txt");
                    StreamWriter TextLogFile = LogFile.CreateText();
```

Now we create a new text file called `SteamApps.txt`, as we don't define a path, then this is created in the current directory, where our application is running from. You could specify a full path if you want the file to be saved somewhere in particular.

```
                    uint []RawAppIDs;
                    RawAppIDs = new uint[AppStats.NumApps];
```

So, let's create an array of the correct type (`uint`) to store our list of AppIDs, notice how we size the array to the number of applications returned by `SteamGetAppStats`.

```
                    iSteamReturn =
UCLSteamEmu.SteamGetAppIds(RawAppIDs, AppStats.NumApps, ref SErr);
```

The call above to `SteamGetAppIds` populates the array we just created with all of the Application IDs from the clientregistry.blob file.

```
                    uint iBuffSize = 512;
```

Now we define a variable to hold a general size value for some strings we are going to create and pass to our external functions. As we don't know what the maximum length should be, we're guessing at 512, and if that crashed, we could up the value a bit. 512 should be more than enough for now though.

```
                    foreach (uint RawAppID in RawAppIDs)
                    {
```

Ok, let's start looping through all the application Ids that are in our array.

```
                        uint iIsAppSubscribed = 0;
                        uint iIsAppReady = 0;
```

First we can get the subscription and ready status of the AppID, so let's create these two variables to pass by reference, which will reveal the status of the application to us.

```
                        iSteamReturn =
UCLSteamEmu.SteamIsAppSubscribed(RawAppID, ref iIsAppSubscribed, ref
iIsAppReady, ref SErr);
```

That is the call to `SteamIsAppSubscribed` that will populate the values for `iIsAppSubscribed` and `iIsAppReady` from the external function.

```
                        UCLSteamEmu.TSteamApp AppNFO = new
UCLSteamEmu.TSteamApp();
```

Great, let's see what else we can find out about this AppID, above we have created a `TSteamApp` object to do this.

```
                        AppNFO.Name = new String(' ', 255);
                        AppNFO.maxNameChars = 255;
                        AppNFO.currentVersionLabel = new String(' ',
255);
                        AppNFO.maxCurrentVersionLabelChars = 255;
                        AppNFO.latestVersionLabel = new String(' ',
255);
                        AppNFO.maxLatestVersionLabelChars = 255;
                        AppNFO.cacheFile = new String(' ', 255);
                        AppNFO.maxCacheFileChars = 255;
```

First we need to initialise it, this is not always necessary for types, but is usually required if we are passing strings and need to define a maximum size for them. Notice how I have populated all the strings with 255 space characters. This will avoid having memory write protection errors if the external function tries to create a string that is longer than the default blank initialised string, i.e. almost anything!

The other way to do this would be to marshal the actual type declaration, but we don't need to, and it's more complex, so we're not going to!

Incidentally, if you run the code without initialising the object first, you will see that it will either crash, or always return defaults.

```
                        iSteamReturn =
UCLSteamEmu.SteamEnumerateApp(RawAppID, ref AppNFO, ref SErr);
```

The call to `SteamEnumerateApp` populates our `AppNFO` object.

```
                    if (AppNFO.numDependencies != 0
                     {
```

This line allows us to cut out some of the unnecessary entries in the clientregistry.blob file and only process what we shall refer to as “valid entries”. There are probably different, and maybe even better ways to do this, but this is what we know for now. Incidentally if we didn’t do this, then you would get applications returned for every possible localisation of a given AppID, which is not good!

```
                        UCLSteamEmu.TSteamAppLaunchOption AppLO =
new UCLSteamEmu.TSteamAppLaunchOption();
```

Now we create a `TSteamAppLaunchOption` object to look at the launch options associated with the AppID.

```
                        AppLO.description = new String(' ', 255);
                        AppLO.maxDescChars = 255;
                        AppLO.cmdLine = new String(' ', 255);
                        AppLO.maxCmdLineChars = 255;
                        AppLO.index = 0;
                        AppLO.iconIndex = 0;
                        AppLO.noDesktopShortcut = 0;
                        AppLO.noStartMenuShortcut = 0;
                        AppLO.isLongRunningUnattended = 0;
```

Again, as it has strings, once more we initialise it.

```
                        iSteamReturn =
UCLSteamEmu.SteamEnumerateAppLaunchOption(RawAppID, 0, ref AppLO, ref
SErr);
```

And here we get a populated `AppLO` object from the `SteamEnumerateAppLaunchOption` function.

```
                        StringBuilder strAppDir = new
StringBuilder((int)iBuffSize);
```

Ah, a `StringBuilder`! These are extremely useful for passing and returning whole strings of a defined size to external functions. Here you can see that we size it to the value we set in `iBuffSize` earlier. Because `StringBuilder` expects its size to be defined as an `int` and `iBuffSize` actually is a `uint` then we precede `iBuffSize` with `(int)` to convert it as required.

```
                            iSteamReturn =
UCLSteamEmu.SteamGetAppDir(RawAppID, strAppDir, ref SErr);
```

Now we can read the installation directory for the given AppID. Note that because `strAppDir` is a `StringBuilder` then we would need to do `strAppDir.ToString()` to actually read its string value at a later date. Note also that you do not need to precede a `StringBuilder` with the `ref` keyword, even though it is essentially being used in a referenced call.

```
                            StringBuilder GCFList = new
StringBuilder((int)iBuffSize);
```

Another `StringBuilder` to hold the list of GCF list returned by hCUPa's `SteamEmu_GetGcfList` function.

```
                            iSteamReturn =
UCLSteamEmu.SteamEmu_GetGcfList(RawAppID, GCFList, iBuffSize);
```

And here we've made the call. Notice this function expects the size of the passed `StringBuilder GCFList` to match the size defined in `iBuffSize`.

```
                            TextLogFile.WriteLine("Steam AppID:      "
+ RawAppID);
                            TextLogFile.WriteLine("      Subscribed: "
+ iIsAppSubscribed);
                            TextLogFile.WriteLine("      Name:       "
+ AppNFO.Name);
                            TextLogFile.WriteLine("      Path:       "
+ strAppDir.ToString());
                            TextLogFile.WriteLine("      cmdLine:    "
+ AppLO.cmdLine);
                            TextLogFile.WriteLine("      GCFList:    "
+ GCFList.ToString());
                            TextLogFile.Write(TextLogFile.NewLine);

                        }

                    }
                    TextLogFile.Close() ;
```

To prove it all works, we write some of the information we have gathered to our `SteamApps.txt` file.

The text file is closed after we have enumerated all the AppIDs.

```
                }
                else
                {

                    MessageBox.Show("Could not find any Steam content
to load!");
                }
```

```
        }else{
            MessageBox.Show("Could not set the Steam User!");
        }
```

Finally, we feedback to the user if there are no Steam applications found in the clientregistry.blob file, or if the Steam user could not be correctly set.

After putting in that lot, run UCL and you should see your usual menu. Quit UCL, then go to the bin\Debug folder within your UCL project folder and look for the file `SteamApps.txt` or browse to whatever location you specified for this file.

Is it there? It should be!

Read through it, you will need to be fully armed for the next tutorial, and if you're really observant, you will notice some problems with the generated file that we will have to work around at a later date.

In the next tutorial, we will look at how we can extend our UCL Content class to accommodate Steam Content, and look into how we can store it efficiently and ultimately use it to launch Steam Content.